FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

ASSIGNMENT OF BACHELOR'S THESIS

Title:	Interactive application to analyze clinical data of patients after transplantation of hematopoiesis
Student:	Allan Kálnay
Supervisor:	Ing. Josef Vogel, CSc.
Study Programme:	Informatics
Study Branch:	Web and Software Engineering
Department:	Department of Software Engineering
Validity:	Until the end of summer semester 2018/19

Instructions

Data of patients who undergo allogeneic hematopoietic transplantation in Institute of Hematology and Blood Transfusion (Ústav hematologie a krevní transfuze - ÚHKT) are stored in MySQL database. Interface for retrieving these data is written in PHP and this interface runs over ÚHKT intranet.

Within the scope of this thesis is to design and implement an interactive web application in programming language R with help of its framework Shiny (shiny.rstudio.com). The practical use of this application is to analyse and visualize data stored in the database. Its outputs will be used for quality monitoring and scientific purposes.

References

Will be provided by the supervisor.

Ing. Michal Valenta, Ph.D.
Head of Department

doc. RNDr. Ing. Marcel Jiřina, Ph.D.
Dean

Prague January 6, 2018

FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

Bachelor's thesis

Interactive application to analyze clinical data of patients after transplantation of hematopoiesis

Allan Kálnay

Department of Software Engineering
Supervisor: Ing. Josef Vogel, CSc.

May 14, 2018

Acknowledgements

Deep acknowledgement goes to my supervisor Ing. Josef Vogel, CSc. who was my main consultant about the thesis' structure. Another one goes to MUDr. Jan Vydra for his patience and willingness to discuss the description with me. Thanks to all the teachers of CTU for their lectures and for all knowledge they have given me. At last, I want to express my sincere gratitude to my awesome family and close relatives for their support during my university studies.

Declaration

I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis.

I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No. 121/2000 Coll., the Copyright Act, as amended, in particular that the Czech Technical University in Prague has the right to conclude a license agreement on the utilization of this thesis as school work under the provisions of Article 60(1) of the Act.

In Prague on May 14, 2018 ……

Czech Technical University in Prague
Faculty of Information Technology

Citation of this thesis

Kálnay, Allan. *Interactive application to analyze clinical data of patients after transplantation of hematopoiesis.* Bachelor's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2018.

Abstrakt

Cieľom tejto práce je vytvorenie interaktívnej webovej aplikácie. Venuje sa analýze a vizualizácii dát o pacientoch, ktorí podstupujú alogénnu transplantáciu krvotvorby. Vizualizácie a analýza sú rozdelené do 3 kategórií a každá z nich sa zaoberá rozdielnou problematikou. Práca detailnejšie popisuje, ako jednotlivé dôležité štatistické elementy, ktoré sú v aplikácii využívané, fungujú. Aplikácia poskytuje uživateľské rozhranie a serverovú logiku, ktoré sú napísané v jazyku R za pomoci frameworku Shiny. Spracovávané dáta sú zhromažďované v databázi MySQL, ktorá beží v lokálnej sieti ÚHKT. Táto aplikácia má slúžiť pre kontrolu kvality vykonávaných úkonov v ÚHKT a pre vedecké účely.

Klíčová slova transplantácia krvotvorby, R, Shiny, štatistika, dátová analýza, dátové vedy

Abstract

The goal of this thesis is to create an interactive web application. The application analysis and visualizes patients data. These patients undergo allogeneic hematopoiesis transplantation. Visualizations and analysis are divided into 3 different categories and each category follows up different problems. The thesis describes in detail, how individual statistical elements used in the application work. The application provides a user interface and server logic. Both of these are written in programming language R with help of its framework Shiny. Processed data are being stored in MySQL database in ÚHKT's local network. This application will serve ÚHKT staff for quality monitoring and it will be also used for scientific purposes.

Keywords hematopoiesis transplantation, R, Shiny, statistics, data analysis, data science

Contents

Introduction **1**

Goals of the work **3**

1 State-of-the-art **5**

2 Theoretical part **7**
2.1 Plots 7

3 Analysis and design **11**
3.1 Analysis 11
3.2 Design 17

4 Realisation **27**
4.1 Implementation 27
4.2 Testing 30
4.3 Deployment 35

Conclusion **37**
Summary 37
Future development 38

Bibliography **39**

A Acronyms **41**

B My outputs **43**

C Installation guide **49**

D Contents of enclosed CD **51**

List of Figures

2.1 Example of Bar Chart 8
2.2 Example of Stacked Bar Chart 8
2.3 Example of 100% Stacked Bar Chart 9
2.4 Example of Scatter Plot with famous Iris data set 9
2.5 Example of Faceting 10

3.1 Iterative development model 12
3.2 Tables *transplantace* and *hctci* from the ÚHKT database 18
3.3 Sketch of the tab Trends in time 21
3.4 Sketch of the tab Frequency tables – old version 21
3.5 Sketch of the tab Frequency tables – current version 22
3.6 Sketch of the tab Survival Analysis 22

4.1 Example of manual testing 33
4.2 Frequency tables tab before the acceptance test 36
4.3 Frequency tables tab after the acceptance test 36

B.1 Transplantations by year 43
B.2 Transplantations by diagnosis 44
B.3 Transplant type 44
B.4 Indications for HCT by year 45
B.5 Transplant type by year 45
B.6 Type of graft 46
B.7 Conditioning regimens 46
B.8 Age of the patients 47
B.9 HCTCI zones 47
B.10 Sex of the patients by year 48
B.11 Survival analysis 48

Introduction

Institute of Hematology and Blood Transfusion (Ústav hematologie a krevní transfuze; acronym ÚHKT) is known for performing hematopoietic cell transplantations for the longest period of time in the whole Czech Republic. Doctors from this institute perform multiple tens of these transplantations in one year and in total, they have performed more than one thousand of them. My work should somehow contribute to the improvement of the methods which they use. Data of each patient who undergo hematopoietic cell transplantation are stored in the database of ÚHKT. The purpose of my application is to analyse these stored data and dynamically create statistics above these data.

When the one sees raw data, many times he can not say much about them. Many times, if not most of the times, raw data are not in appropriate form. They are not adjusted for further manipulation. There exist different kinds of problems for raw data but, fortunately, the data that my application analyses have very good shape and mostly it is not necessary to anyhow clean these data.

Statistics and analyses created by my application are divided into 3 categories. Each one of these categories deals with different problem(s). The 3 categories are:

1. **Trends in time**: In this category, I examine the evolution of attributes of my dataset in time. Mostly this is a matter of visualizing attributes of the dataset by time period.

2. **Frequency tables**: This tab contains an external application that I have integrated into my application. Its capability is to plot frequency table which displays the frequencies of all the values which the variables take on. Moreover, this external application is able to perform some other statistical operations over data like plotting or computing statistical functions.

3. **Survival analysis**: This one plots Kaplan-Meier curves on graphs. These curves analyse survival in time. Each point on curve describes survival probability till the exact moment.

Theoretical part names and describes the types of plots that I use in my application.

The practical part of the work contains the analysis of the problem, design of the application, its implementation and in the end testing.

Front-end and server logic is written in R and its package Shiny and the application retrieves data from ÚHKT's MySQL database.

Goals of the work

The goal of my work is to design and implement an interactive web application for visualizing and analysing data of patients. This application is meant to help the ÚHKT staff to easily read these data and get some knowledge out of them. The application will help ÚHKT staff with quality monitoring and it will be also used for scientific purposes.

Chapter 1

State-of-the-art

Till this moment the ÚHKT staff was dealing with the problem of analysing data by generating PDF documents 4 times a year. They have developed R scripts which have always done the job. It is easily understandable that generating the documents is useful for different kind of analysis than using an application that is able to dynamically generate plots. Moreover, not everybody could have created these statistics but only those who were capable of such things – the people who know R and who know how to work with these scripts. So my application will allow users to check the statistics without any further skills from the field of information technologies.

Dynamic data plotting have become a completely new problem domain for ÚHKT and my application is opening the first chapter for them in this field. Now, ÚHKT is the collector of patients data which are going to be used by my application. They have the database for storing the data and they have the web application that provides the UI allowing users to insert data into the database.

Chapter 2

Theoretical part

2.1 Plots

"*A plot is a graphical technique for representing a data set, usually as a graph showing the relationship between two or more variables.*" [2]

Plot, chart or graph – whatever word of these 3 I use in this text, I mean the same, the graphical representation of data. If I want to be more concrete, **plot** is the superset of **chart/graph**.

There exist many different types of plots. Some of them are more popular, others are less popular. Most of the plots used by me in my application are the more popular ones. Let's describe the plots that I use in my thesis and in my application.

(Simple) Bar Chart
We will focus only on 2D bar charts because those are the ones that I use in my work. Bar chart is the type of graph that is used to display counts of categorical data. Bar chart uses Cartesian coordinates. One axis represents a categorical variable and the other axis represents the counts of the categories of the categorical variable. There exists a bar for each category. Each bar represents different category and the height of the bar represents the absolute count of data in the category. A lot of my visualizations are exactly of this type. Below is an example of simple bar chart. Figure 2.1 is the example showing absolute counts of **butterfly species** in some observation. You can see that the bars are displayed vertically. These bars can also be displayed horizontally (flipped axes). It is very common to display the bars vertically but it depends on a person who creates the plot whether he displays the bars vertically or horizontally.

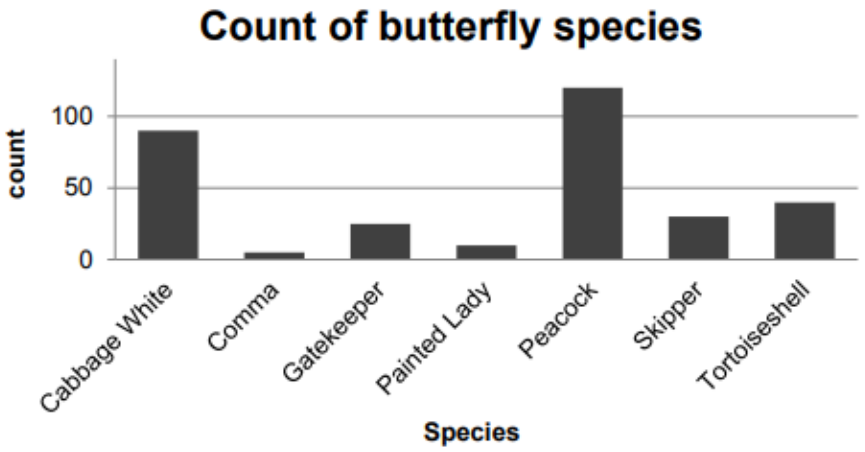

Figure 2.1: Example of Bar Chart; source: `https://portal.uea.ac.uk/documents/6207125/8203361/steps+into+statistics+bar+charts.pdf`

Stacked Bar Chart

This type of chart is similar to Simple Bar Chart described above. Stacked Bar Chart divides data into categories by a categorical variable as Simple Bar Chart does and on top of that, this chart type divides each column into several groups based on another variable. The counts can be displayed absolutely as in figure 2.2 or relatively (100% Stacked Bar Chart) as in figure 2.3. In this context, *relatively* means that the chart displays the counts relatively to each group represented by a single bar – in percentages – and this chart is similar to well known Pie Chart.

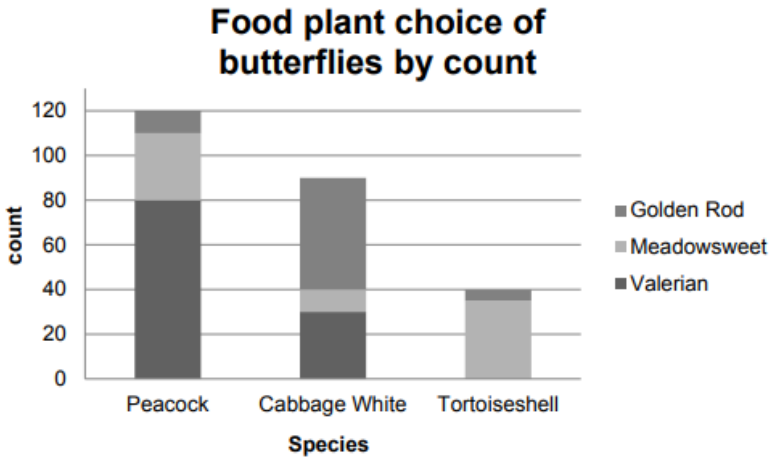

Figure 2.2: Example of Stacked Bar Chart; source: `https://portal.uea.ac.uk/documents/6207125/8203361/steps+into+statistics+bar+charts.pdf`

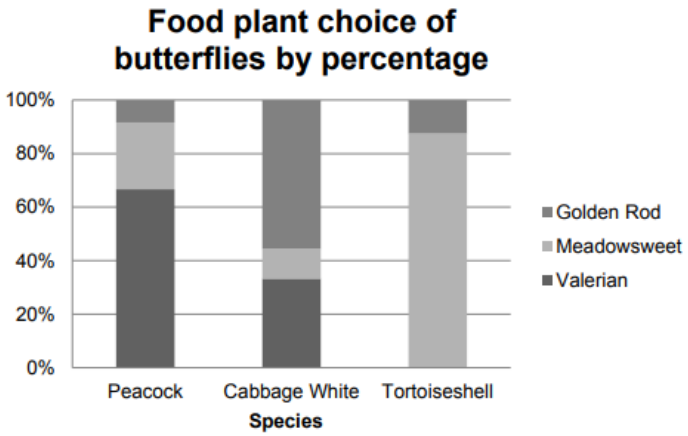

Figure 2.3: Example of 100% Stacked Bar Chart; source: `https://portal.uea.ac.uk/documents/6207125/8203361/steps+into+statistics+bar+charts.pdf`

Scatter Plot

Scatter Plot is a plot using Cartesian coordinates that typically displays the relationship between 2 or 3 quantitative variables. One more categorical variable can be added into a plot by colouring the points based on the category they belong to. Example figure 2.4 shows the relationship between *Sepal.length* and *Sepal.width* attributes. Additionally, data of the same category of attribute *Species* are coloured with the same colour.

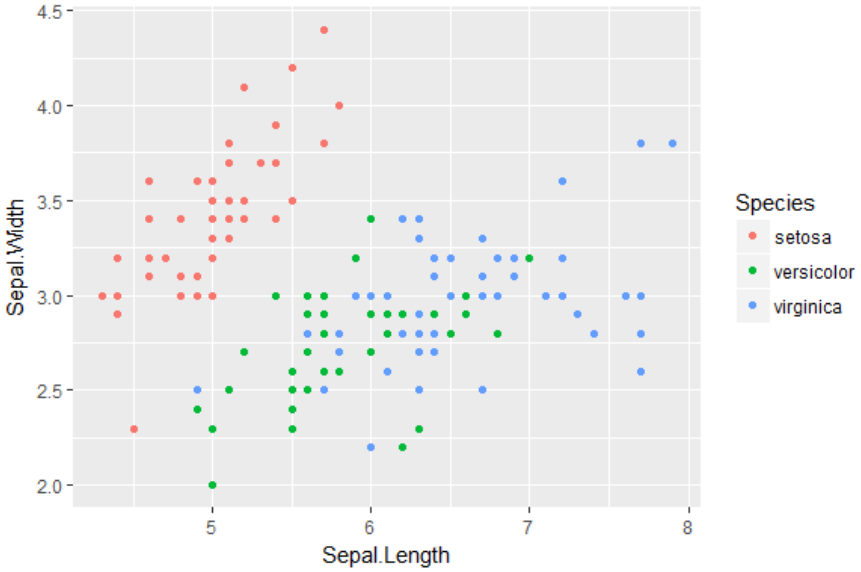

Figure 2.4: Example of Scatter Plot with famous Iris data set

Facet

Facet partitions a plot into a matrix of panels defined by column and/or row faceting variable. An image is worth thousands of words:

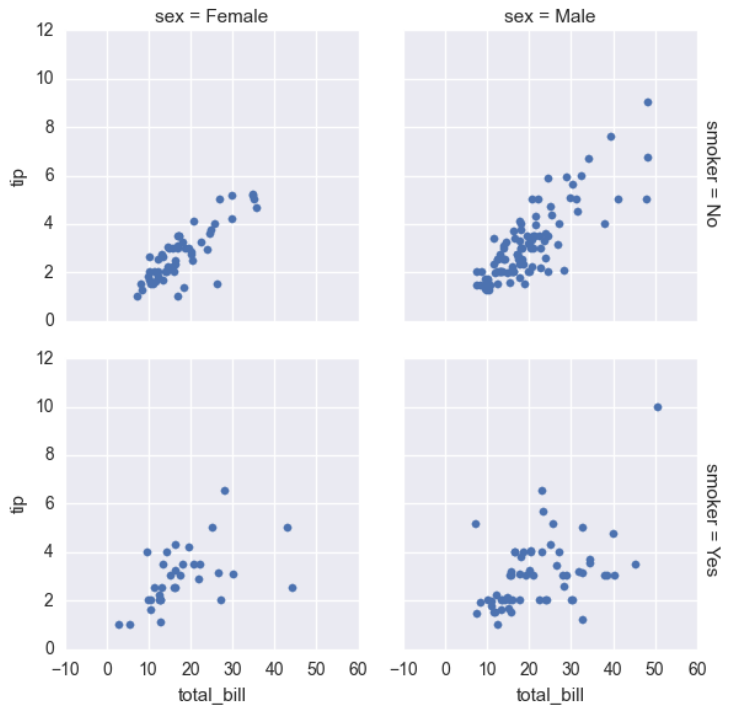

Figure 2.5: Example of Faceting; source: `https://i.stack.imgur.com/Bk0na.png`

The column faceting variable of the example is *sex* and the row faceting variable of the example is *smoker*.

Chapter 3

Analysis and design

Before I started writing the application itself, it was necessary to break down the problem. Since the application will be many times extended if the ÚHKT staff decides to do so, the problem needs very patient designing phase. This application is not only ready to analyse patients attributes but is ready to analyse anything else. Any other databases can be linked with my application and analyses of these additional data may be added.

I have to emphasize that the UI of the application is written in Czech. In this text, I will use English language in the first place but I will also provide translations into Czech. These Czech phrases are used in the application's UI.

3.1 Analysis

The intention of system analyses is to describe what is really going on in the problem domain. When I started the communication with ÚHKT, they already had quite a good idea how the application should look like and how it should work. So I have sat down with my client, he has described me the problem and the requirements and I have written it down.

The key of my whole development process is the chosen iterative development approach. Since they have had an idea how the application should look like, iterative development has been the best option. Figure 3.1 captures iterative development approach.

3.1.1 Why not to use existing solutions

There are some existing analytical programs that somehow solve some parts of the problem but it will not provide the ÚHKT staff with the option to build up further upgrades when the practice calls for them. Programming language R ideally solves this kind of problem – R provides all the medical statistical features which ÚHKT needs. Also, my solution of the problem provides the ÚHKT staff the option to add to the application anything they want. In fact,

R does not limit them in any way. The application may be extended in any client's desired way. ÚHKT staff is sure that R will provide them everything they need.

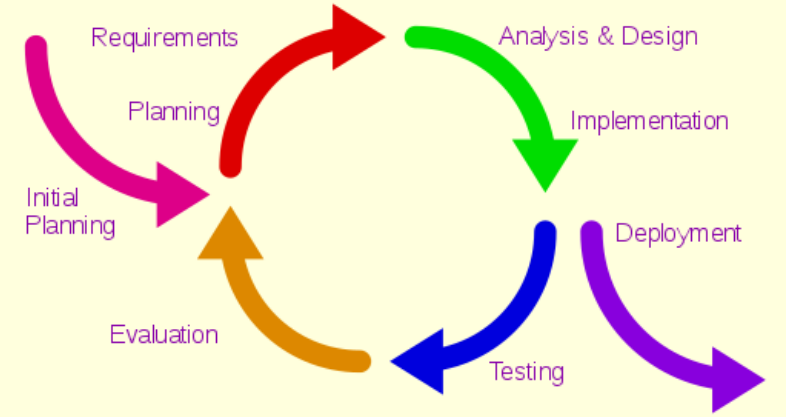

Figure 3.1: Iterative development model; source: `https://en.wikipedia.org/wiki/Iterative_and_incremental_development`

3.1.2 Requirements analysis

Following instructions of software development process written in [3], requirements analysis is one part of the software analysis. Requirements analysis helps both sides – the client and the distributor – with the assignment clarification. It gives a closer look at how the system should work.

From the first to the last iteration I have written down all the following functional and non-functional requirements with my client.

3.1.2.1 Functional requirements

- F1 – the application will visualize and analyse data in named plots,
- F2 – the application will allow users to filter certain plot outputs,
- F3 – the application will plot survival curves,
- F4 – the application will display frequency tables using an integrated external application.

Description of functional requirements

F1 The main purpose of the whole application is to visualize data. So the application will plot specified graphs out of the specified dataset. Each plot in my application has its name except few that are described in other functional requirements. The application has to contain the following named plots:

  - *Transplantations by year*,
  - *Transplantations by diagnosis*,
  - *Indications for HCT by year*,
  - *Transplant type*,
  - *Transplant type by year*,
  - *Type of graft*,
  - *Sex of the patients by year*,
  - *Age of the patients*,
  - *HCTCI zones*,
  - *Conditioning regimens*,
  - *Conditioning regimens used by year*,

F2 Some plots are static – if a user visits my web application, these plots get loaded and that's all. Other plots are dynamic – if a user visits my web application, these plots get loaded, their input panels get loaded and these input panels will provide filtering data shown in plots.

F3 The application will contain a plot of survival curves. This plot does not have its name as the plots in F1 have so this functional requirement is separated from F1.

F4 The application will plot frequency tables in the tab named Frequency tables. An external application described later in Analysis section will provide this feature. This application will be integrated into my application.

3.1.2.2 Non-functional requirements

- N1 – the application will be the web application,
- N2 – visualized data will be loaded from ÚHKT's MySQL database,
- N3 – plots' loading time will be as short as possible,
- N4 – programming language is R and web framework is Shiny,
- N5 – easily readable code prepared for extensibility and further improvements,
- N6 – friendly and intuitive UI,
- N7 – the application will contain 3 tabs (menu).

Description of non-functional requirements

N1 The application is meant to be the web application. The application is not required to be adjusted for devices like tablets, mobile phones et cetera. Anyway, Shiny does provide does provide responsivity because of its usage of Bootstrap 2.

N2 The ÚHKT's database of patients data will be the data source of my application.

N3 User should not wait for graphs for a long time. This performance requirement is very key one because it is always uncomfortable for a user to wait for a long time for the desired output.

N4 ÚHKT staff has to be able to extend the code if it is needed. They know R and that is why it is needed to use this language. Shiny is a package of R so it is not a problem to get into it quickly and it is the best way to create R web applications.

N5 This requirement goes hand in hand with requirement N4. Besides that the code's architecture has to be well designed for easy further manipulation.

N6 The charts which demand legend will have legend by its side for an explanation. Axes will be adequately labelled with values as well. Graphs will be titled.

N7 The application's GUI will contain menu on the very top of the application's border. This menu will contain 3 tabs – **Trends in time**, **Frequency tables**, **Survival analysis**. Each of these tabs will analyse different topic.
Trends in time tab will contain graphs displaying occurrences of certain values in time periods (in years).
Frequency tables tab will contain the external integrated application that solves the problem of displaying frequency tables.
Survival analysis tab will visualize survival analysis curves.

3.1.3 Use cases

The one and only known use case is **Data analysis** and the only actor is **Guest**.

Guest may be anyone who can visit the application's web page. Mostly these will be doctors but it may also be a nurse, a cleaner, a medical student,...

Data analysis use case says that Guest may perform data analysis using application's plots and its filters.

3.1.4 Technologies

This section breaks down the main technologies which I have used to develop this project and describes why I have used them. The main technologies were decided in the analysing phase. All the other ”side” technologies were not known in analysis phase and they will be described in chapter 4.

3.1.4.1 Why R

R is a programming language that was created for statisticians and statistical purposes. For a programmer that is used to languages like Java, C, C++ or any other similar languages, R may be a bit confusing. In the end, R is very intuitive and it has all you need when it comes to statistics. I have not run into a situation when I needed to implement any statistical algorithms while using R yet.

The fact that R is the well-known language for my client was the main reason for choosing R as the programming language. Another main reason why I have used R is that R provides a programmer all the statistical features that ÚHKT needs. These guys have to be able to extend the application in the future. The code has to be easily readable. The code has to be intuitive. R offers all this stuff.

An application written in R can run on any operating system which has installed interpreter for R. So it does not matter whether the application will run on Windows, Linux, macOS or anything else that R can be installed on. If the R interpreter is present and all the dependencies (packages) are present as well, the application will run.

3.1.4.2 Why Shiny

This application is completely data scientifically oriented. It does not require anything else than taking input from a user and based on this input plot some graphs or do other statistics.

Shiny was built for developing standalone web applications and is relatively easy to use. It does not require any knowledge of HTML, CSS neither JavaScript. Anyway, it is good to know these because it may help you in some situations.

Shiny also does not require any external software like Apache to be installed on the server except for R, Shiny and packages that are included in an application. Shiny does all the work for you when it comes to making a server out of your computer.

3.1.4.3 RStudio

For the development of this application, I have used IDE called RStudio. It is the open-source and user-friendly IDE used for developing R applications.

It is available for Windows, Linux and macOS.

RStudio especially helps programmer with the following things: syntax highlighting, code completion, easy file management, project creation, integrated R documentation, easy export of graphical outputs to PDF, PNG and some other file formats.

3.1.4.4 External application

Shiny application by creators of compareGroups package [6]
One part of the application consists of displaying frequency table – tab called Frequency tables. A user chooses the names of the dataset's variables which he wants to be visualized in the table and the application will do so. The first idea about the functionality of this frequency table and its first realisation was showing a user the frequencies of all the combinations of values of user-chosen variables. After some time my client has found R package called **compareGroups** that displays frequency table very user friendly[1]. Creators of this package have developed the Shiny application using this package that does exactly what my client needs my application to do – to display the frequency table – and in addition to that it has some other features that will help him with analysis. So he has decided to let me integrate this application into my application. When I mention the phrase the ***external application*** later in this text, I am referring to the application that I have just described in this subsection.

First implementation versus external application
At first, I and the client have agreed on displaying a contingency table in the tab Frequency tables. Design and the implementation of this old version of the tab Frequency tables is closely described in this chapter, in the section Design, and in the chapter Realisation.

There was no further analysis on the topic the external application. The client has decided that this external application will be more useful for him due to the following reasons:

- compareGroups package allows creating frequency tables for single attributes. The first solution allowed deeper analysis (for more variables at one time), however, in most cases it is not needed.
- Additionally, compareGroups displays basic statistics for dataset's variables – mean, standard deviation, median, dispersion, p-Value and few more.

Due to the mentioned reasons, the external application is an appropriate statistical tool for the ÚHKT staff.

[1] This application does not work exactly same way as the old implementation worked. Soon I will explain everything.

3.1.4.5 MySQL database

MySQL is an open-source relational database management system. MySQL database has already been running over the ÚHKT intranet before I started developing this application. Data which my application will work with are stored in this database. So I have not done any analysis about databases.

3.2 Design

The application has 3 tabs. Each tab represents different page and each tab analyses data from different topics. Whatever tab a user visits, the application will display the page of this tab and the page will contain menu at the very top of the page with these 3 tabs. The content of each page is different and it will be discussed soon.

3.2.1 Database of ÚHKT

ÚHKT runs MySQL database over their internal network. This database contains 57 tables but not all of them are important for my application. In fact, my application needs just very few tables from this database and very few columns. Here, I will describe what data my application works with and how.

One of the most important data for my application are from table *transplantace*. This table contains information about patients who undergo transplantation. This table contains 50 columns in total but not all of them are really important for me. Another relevant table is *hctci*. My application uses only 1 column of this table, **hctci_score**.

Figure 3.2 displays relevant tables for my application and its columns. Column *id* of *transplantace* table represents transplantion's ID (*transplantace.id* equals to *hctci.id*).

Let's describe the relevant columns of tables *transplantace* and *hctci*:

- **vek** represents the age of the observed patient.
- **pohlavi** represents the sex of the observed patient.
- **dg_group** represents the patient's diagnostic group.
- **dg_dat** represents the date when the patient's diagnosis was set.
- **tx_dat** represents the date when the patient's transplantation was performed.
- **pacient_zemrel** represents whether the patient has died after transplantation or not.

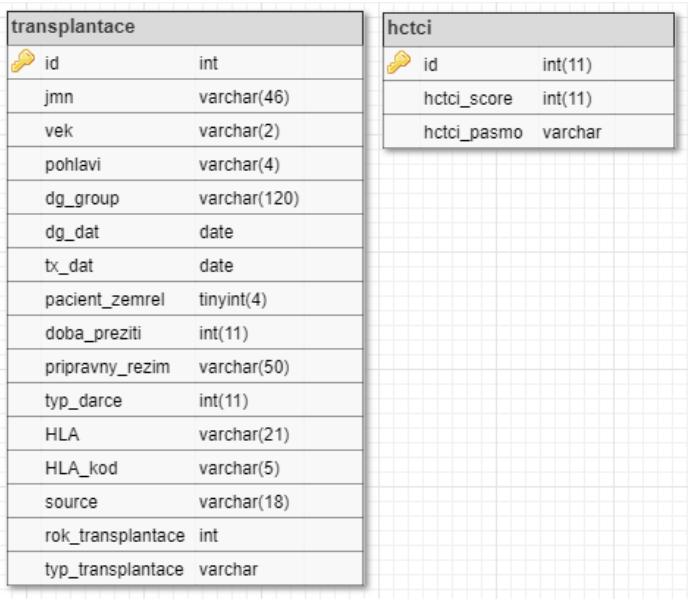

Figure 3.2: Tables *transplantace* and *hctci* from ÚHKT's database – this image contains only relevant columns for my application

- **doba_preziti** represents how long the patient lived after the transplantation.
- **pripravny_rezim** represents conditioning regimen which the patient has undergone. It is a chemotherapy (sometimes a radiotherapy) which the patient undergoes for 7 days before grafting. Its goal is to allow the donee to accept the donor's cells. Another goal of conditioning regimen is to clean the cancer cells in the patient's organism.
- **typ_darce** represents the type of the patient's donor (e.g. sibling, unrelated, relative,…).
- **HLA** represents patient's human leukocyte antigen.
- **HLA_kod** expresses how coincident is the patient (the donee) with his donor in 10 examined HLA genes. The value 10/10 means the coincidence in all HLA genes. The value 5/10 means the coincidence in the half of the HLA genes.
- **source** – this variable is called **typ_stepu** in my dataset. It represents the type of the patient's graft.
- **hctci_score** represents HCT-CI. "*HCT-CI is a comorbidity index that comprises 17 different categories of organ dysfunction.*" [4]

There are 2 more attributes named **rok_transplantace** (the year of the transplantation) and **typ_transplantace** (transplant type) sketched in the table *transplantace*. In fact, it is not part of this table and it was artificially added. These variables are just part of my dataset and they are derived from the table *transplantace*.

The 1st attribute, **rok_transplantace**, is part of my dataset and I get it by SQL's `EXTRACT()` function. This attribute is just the year extracted from *transplantace.tx_dat* column. It represents the year when the patient underwent the transplantation.

The 2nd one, **typ_transplantace**, is also part of my dataset. The 4 other attributes decide what type of transplantation the patient has undergone. These are: *transplantace.typ_darce*, *transplantace.HLA_kod*, *transplantace.HLA*, *transplantace.typ_stepu*[2].

The table *hctci* contains one more attribute, **hctci_pasmo**, that was not mentioned yet because it is not part of the table. This attribute is also artificially added. It is derived from *hctci.hctci_score*. This attribute represents HCT-CI zones and it takes on one of the 3 values: **0** ($hctci\_score = 0$), **1-2** ($hctci\_score \in [1, 2]$), **>2** ($hctci\_score > 2$).

Data from the table *transplantace* are used both in tab Trends in time and tab Survival analysis. Data from the table *hctci* are used only in tab Trends in time.

The next data that my application works with – concretely tab Frequency tables works with these data – are retrieved from ÚHKT's database by the view called *transpl* created by ÚHKT staff. These data contain all the attributes mentioned above plus some others that were not mentioned. These attributes do not necessarily need to be described. The external application works with these data and there is not needed any further analysis of these data. The external application processes them the way ÚHKT staff wants them to be processed.

3.2.2 Design of the graphical user interface

The client has had a rough idea of how the GUI should look like. I have discussed the GUI with the client, he has shown me some sketches of the GUI and I have polished some little pieces to finish it. I will expose how the GUI should look like with wireframes in this section. A wireframe is a 2D sketch of a graphical user interface. It does not contain any graphical elements. It is just an image that shows how the final layout of a web page should approximately look like. I have created my wireframes using a web application for creating wireframes provided by MockFlow[3].

[2]When I want to refer to a specific column of a table, I will express it by the following formula *table_name.column_name*.

[3]https://mockflow.com

Each page has the following structure:

- **header** – this is the place where the menu with the 3 tabs is located,
- **body** – the body of each page is located directly under the header. Body of each page contains the main content of the page – the plots, the titles of the plots, inputs allowing users to filter the outputs of the plots et cetera.

Tab Trends in time
The translation of the name of this tab is *Trendy v čase*. This page is supposed to analyse data in time. The page of this tab contains the header and the body as any other page. The body of this page contains sections and these sections are divided by horizontal lines. Each section has to contain the following elements:

- **the title** of the section (of the analysis),
- **the plot**.

Additionally, some sections may contain a panel with user input elements. Each input element has its label by its side. The following input elements are used on this page:

- double-ended range input (slider) representing the time period,
- select input – for selecting a concrete year.

Tab Frequency tables
The translation of the name of this tab is *Tabulky výskytů hodnot*. Section 3.1.4.4 provides the brief description of how this tab worked the first time it was implemented and how it works after integrating the external application. At first, the body of this page contained the panel with user input elements called *checkboxes* and the frequency table itself. The checkboxes represented the dataset's variables. A user could have chosen variables he wanted to be displayed by checking the relevant checkboxes.

Tab Survival analysis
The translation of the name of this tab is *Analýza přežití*. The body of the page for this tab contains the panel for user input and the chart with survival curves as shown on figure 3.6.

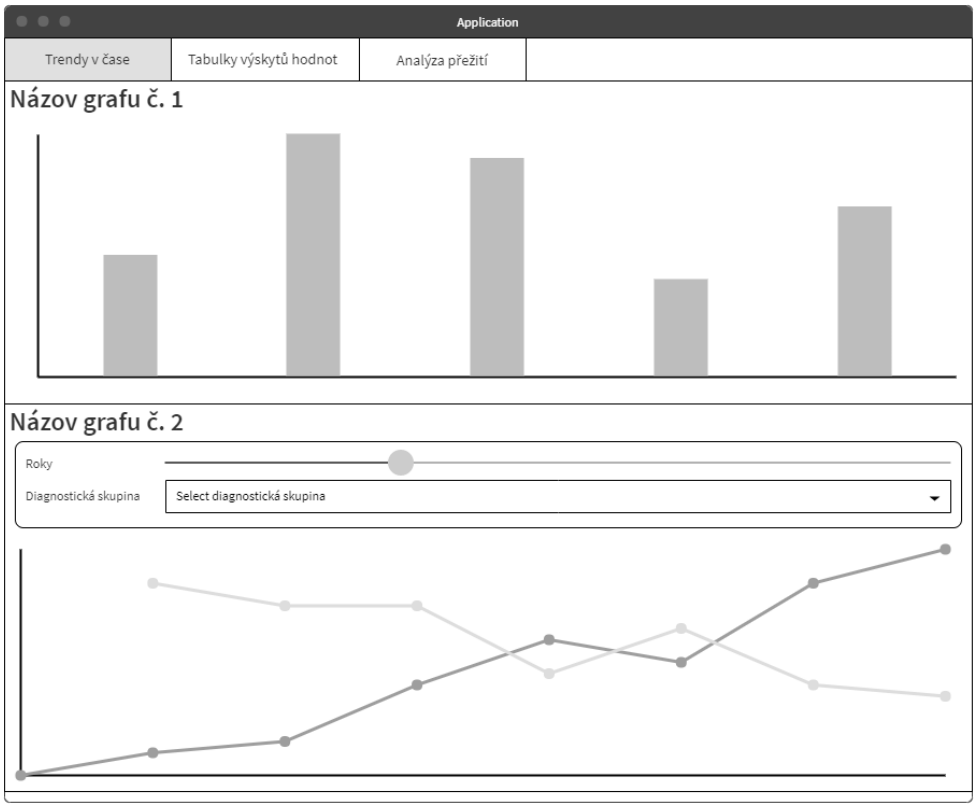

Figure 3.3: Sketch of the tab Trends in time

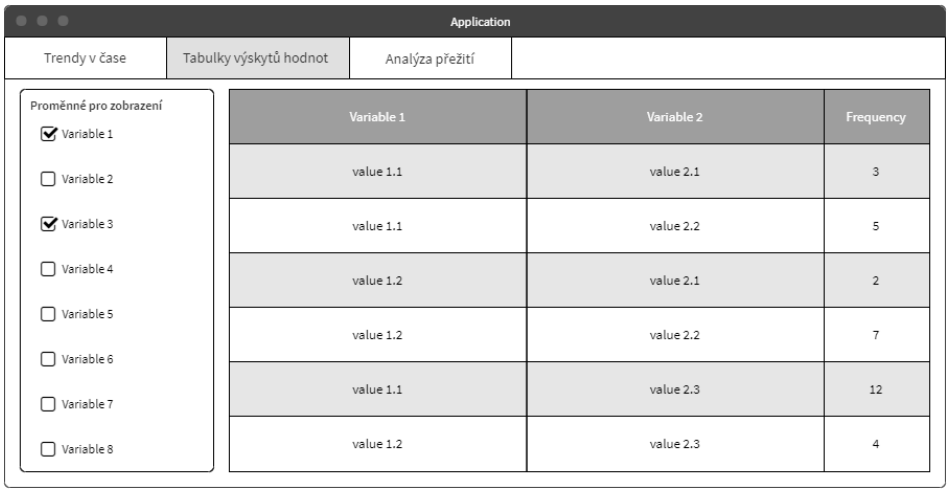

Figure 3.4: Sketch of the tab Frequency tables – old version

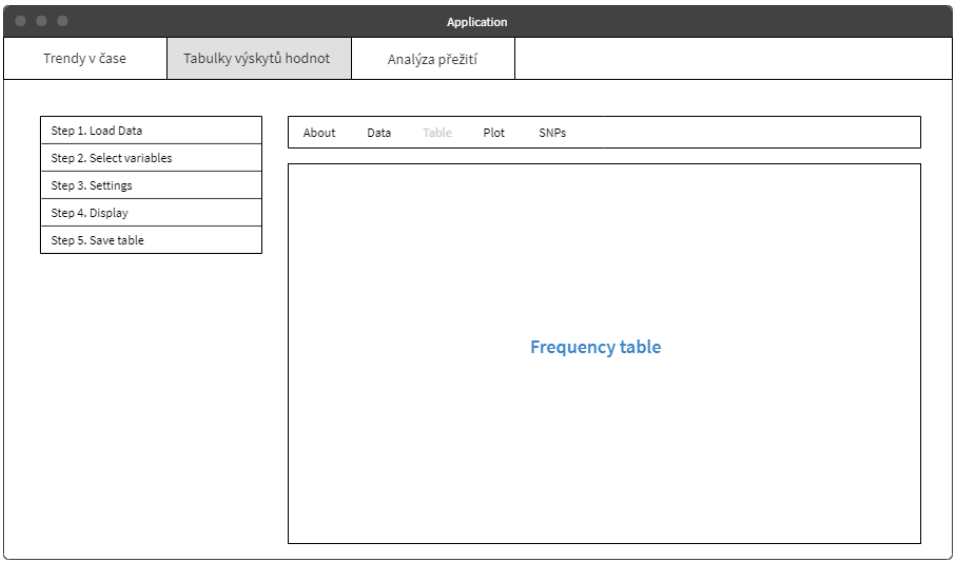

Figure 3.5: Sketch of the tab Frequency tables – current version

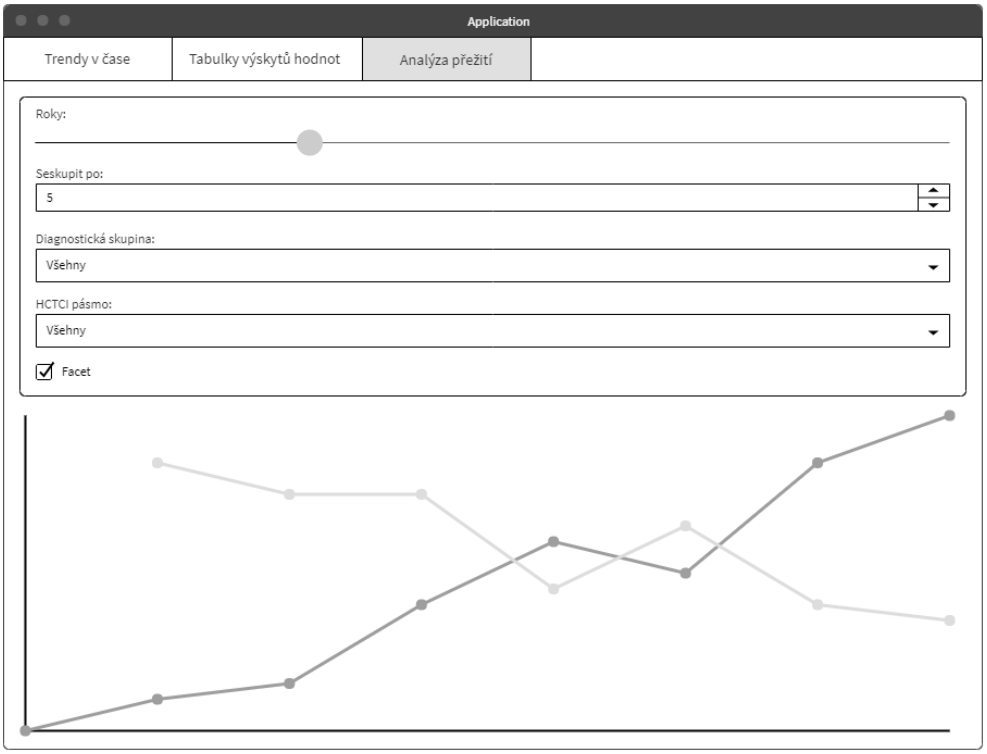

Figure 3.6: Sketch of the tab Survival Analysis

3.2.3 Plotting

This section describes the plots of single tabs. There is written here what types of plots will be displayed, at what positions and what data they will proceed.

3.2.3.1 Tab Trends in time

Some of the following plots may contain user input elements in their user input panels. The following input elements are possible:

1. **select input** for selecting a concrete year named "Rok". This select input contains all the unique values from the variable *transplantace.rok_transplantace* in descending order from top to bottom,

2. **double-ended slider** named "Roky". This slider contains all the unique values from the variable *transplantace.rok_transplantace* in ascending order from left to right.

Transplantations by year
Translated as *Transplantace v letech*. It is a bar chart. x-axis displays categorical variable *transplantace.rok_transplantace*.

Transplantations by diagnosis
Translated as *Transplantace podle diagnóz*. It is a bar chart with flipped axes. y-axis displays categorical variable *transplantace.dg_group*. This plot contains user input number 1.

Indications for HCT by year
In the application's UI it is called *Vývoj spektra diagnóz*. It is a bar chart with flipped axes and faceting. y-axis displays categorical variable *transplantace.dg_group* and the column faceting variable is *transplantace.rok_transplantace*. This plot contains user input number 2.

Transplant type
Translated as *Typ transplantace*. This one is represented by bar chart with flipped axes. y-axis contains categorical variable *transplantace.typ_transplantace*. This plot contains user input number 2.

Transplant type by year
Translated as *Typ transplantace – vývoj v čase*. It is a stacked bar chart. x-axis contains categorical variable *transplantace.rok_transplantace* and the columns are filled with another categorical variable *transplantace.typ_transplantace*. Counts are displayed absolutely.

Type of graft
Translated as *Typ štěpu*. It is displayed as bar chart with flipped axis. y-axis contains categorical variable *transplantace.source* (also called *transplantace.typ_stepu*).

Sex of the patients by year
Translated as *Pohlaví pacienta v jednotlivých letech*. This one is a stacked bar chart with relative values. x-axis contains categorical variable *transplantace.rok_transplantace* and y-axis shows percentage. The bars are filled with categorical variable *transplantace.pohlavi*.

Age of the patients
Translated as *Věk pacientů*. This one is of the scatter plot type. x-axis contains categorical variable *transplantace.rok_transplantace* and y-axis contains the values of continuous variable *transplantace.vek*. Also this plot contains median of *transplantace.vek* for each year – distinguished by red colour.

HCTCI zones
Translated as *HCTCI pásma*. This one is a stacked bar chart with relative values. x-axis contains categorical variable *transplantace.rok_transplantace* and y-axis shows percentage. The bars are filled with categorical variable *transplantace.hctci_pasmo*.

Conditioning regimen
Translated as *Přípravný režim*. This one is represented as a bar chart with flipped axis. y-axis contains categorical variable *transplantace.pripravny_rezim*. This plot contains user input number 1.

Conditioning regimens used by year
Translated as *Vývoj spektra přípravných režimů*. This one is a bar chart with flipped axes with faceting. y-axis displays categorical variable *transplantace.pripravny_rezim* and the column faceting variable is *transplantace.rok_transplantace*. This plot contains user input number 2.

3.2.3.2 Tab Frequency tables

This tab contains the external application. The only thing that needs to be done here is to integrate the application. The whole structure of the external application will stay untouched except reading data system. The original system of reading data has to be replaced with only 1 button *Read data* that will load my dataset.

3.2.3.3 Tab Survival analysis

This tab contains 1 scatter plot displaying survival curves. x-axis represents survival time in months. y-axis represents survival rate in percentages. Each survival curve represents survival analysis of certain time period (in years). The time periods are disjoint, e.g.:

- **curve 1** – time period 2000-2002,
- **curve 2** – time period 2003-2005,
- **curve 3** – time period 2006-2008.

The user input panel contains inputs for filtering the plot output. The inputs are the following:

- **double-ended slider** "Roky:" for choosing a time period (since a year until a year),
- **numeric input** "Seskupit po:" for grouping data by time period (in years),
- **select input** "Diagnostická skupina" for visualizing data from certain diagnostic group,
- **select input** "HCTCI pásmo" for visualizing data of certain HCT-CI zone,
- **checkbox** "Facet" for separating single curves into its own charts. *Checked* separates curves into multiple charts, *unchecked* puts all the curves into a single chart.

3.2.4 Directory layout

One of my designing parts is directory layout. Designing the right directory layout helped me at my implementation process. I have assigned the specific role to each directory and each file of the layout. The following directory tree describes the roles of key files and directories.

- `db` ..... the directory with R scripts operating over DB
- `external` ..... the directory containing external applications
  - `compareGroups` ..... the directory containing external application
- `server` ..... the directory containing R scripts with server logic
- `tests` ..... the directory containing all tests
  - `tests_chart` ..... the directory containing unit tests for charts
- `ui` ..... the directory containing all R scripts forming UI
- `www` ..... the directory with any UI-supportive files, e.g. CSS
- `app.R` ..... main file of the Shiny application
- `constants.R` ..... contains constants used across the whole application
- `dependencies.R` ..... contains application dependencies, e.g. database connection or loading data
- `functions.R` ..... contains R functions used across the whole application
- `options.R` ..... contains R options, e.g. encoding

There exist other files (subdirectories as well) which were not mentioned in the directory tree. It is not worth mentioning them because their functionalities are obvious and their functionalities are basically described by their parent directories in the directory tree above and by their names.

Chapter 4

Realisation

The chapter Realisation is divided into 3 main sections – Implementation, Testing, Deployment. These 3 are the key components of software realisation.

My Implementation section mostly says something about packages that helped me to achieve the goals.

Testing section gives a view on the penultimate and very important part of software development. You can see what testing styles I have applied and why I have applied them.

Finally, Deployment section will inform you about the application's deployment in ÚHKT's places.

4.1 Implementation

4.1.1 The basic structure of Shiny application

Shiny application has two main components – a UI and a server logic. The UI is built by function `shinyUI()` and the server logic is built by function `shinyServer()`. These 2 can be divided among different files but also they can be located in the same file. Since my application is not a small one, I have decided to separate these 2 – the UI and the server logic. So the function creating UI is located in *ui.R* file and the function creating server logic is located in *server.R* file.

This structure would be sufficient if I did not need any other helping files to be included. Since I need these helping files, I need one more file called *app.R* which contains `shinyApp()` function that takes 2 parameters – `ui` created by `shinyUI()` and server logic parameter named `server` created by `shinyServer()`.

When the Shiny application wants to get started, first the system looks for the file called *app.R* in the application's folder. If it is found, this file is loaded and the application is started. Else the system looks for 2 files called *ui.R* and *server.R* and loads these 2. [9]

4.1.2 Beyond the basic Shiny structure

The directory layout is roughly described in 3.2.4. Let's look closer at the important files and directories.

***ui* and *server* directory**
As the names say – the first one contains all R scripts building the UI and the second one contains all R scripts building the server logic. Both of them contain scripts with these names: *trendy_v_case.R*, *tabulky_vyskytu_hodnot.R* and *analyza_preziti.R*. These files either create the UI for the relevant tabs (in the *ui* directory) or create the server logic for the relevant tabs (in the *server* directory.

Beyond the scripts mentioned above, *ui* directory contains the main script named *my_ui.R*. This script builds the whole UI with the mentioned `shinyUI()` function. The other scripts are included in the main script.

The same goes with *server* directory. Its main script is named *my_server.R*. This script builds the whole server logic with the mentioned `shinyServer()` function and the other scripts are included in this script.

For both cases – *ui* and *server* directory – it works the same way – each of the main scripts include the 3 minor scripts mentioned in the 1st paragraph and therefore the whole UI and the server logic are built.

***db* directory**
This directory contains 2 R scripts – *db_connection.R* and *query.R*.

The 1st one, *db_connection.R* represents the core of work with the database. It contains information about database connection (database server name, database name, port, username, password) and the functions for working with database (the function for creating the connection and the function for loading data from database).

query.R script contains the constants holding query strings.

***www* directory**
This directory serves as the repository for CSS files and the images used on the website. There are several bootstrap CSS files in this directory used by the external application, my main CSS file called *styles.css* and the images used by the external application.

***external* directory**
This directory serves as the repository for external entities. The external application is located here as well and it has its own directory *compareGroups*. *compareGroups* directory does not contain the core of the application – it is located in *ui* and *server* directories – but it contains the side and auxiliary files.

***functions.R* script**
This R script is meant to contain all the functions that do not belong to any other section like server logic, UI, database related stuff et cetera. There are mostly functions used for manipulating my dataset.

4.1.3 Data manipulation

R itself provides API for data manipulation but in my opinion, its code is not so transparent as R package **dplyr** is. So for data manipulation, I have chosen dplyr package. dplyr defines couple of functions for data manipulation – `select()`, `filter()`, `mutate()`, `arrange()`, `group_by()`, `summarise()` – and imports the pipe operator `%>%` from another package named magrittr. The pipe operator redirects the output from one function to the input of another function. So the pipe operator provides reading the functions from left to right. [8]

The following example shows how transparent the dplyr code is. It selects *age* and *sex* variables from `my_data` dataset and only the entries of *transplant_year* equals to 2017:

```
my_data %>%
  select(age, sex) %>%
  filter(transplant_year == 2017)
```

4.1.4 Working with database

For working with the database I have decided to use the package called **dbplyr**. I have chosen this package because it allows me to work with remote database tables or the results of SQL queries as if they were in-memory data frames. This means that I do not have to retrieve data from the database via direct SQL queries. dbplyr allows me to create a reference on a table or on any other result of SQL query and work with that like with an in-memory data frame. So I can work with these references the same way as I described in, 4.1.3, the section above and dbplyr makes SQL queries behind the scenes.

4.1.5 Plotting with ggplot2 package

There exist several alternatives for plotting. One of them is ggplot2 package. ggplot2 is R package for creating graphics based on the grammar of graphics. I use ggplot2 in server logic code for creating the plots across the whole application. [5]

4.1.6 Refactoring code of the external application

The only thing needed at integrating the external application was changing the built-in data loading system. Since the external application is the open-

source software I have just refactored the code of data loading system and I have adjusted it for loading my dataset. Everything else has remained untouched. One very good remainder in this section is that this application uses **compareGroups** package. This package is responsible for the logic of the application – descriptive (frequency) tables and the plots.

4.1.7 Plotting survival curves

For plotting survival curves I have chosen multiple packages. Each of them deals with different steps till the final plot is achieved. These packages are described in the following list.

survival is used just for creating survival curves. Attention: this package does not plot the survival curves, it just creates them (only holds some information necessary for later plotting).

survminer contains the function `ggsurvplot` which plots the survival curve(s) created with survival package.

ggplot2 plots the survival curves only in a case if a user wants the faceting to be applied. In this case, the faceting variable is *time period* – for each time period exists only 1 survival curve at single moment.

4.2 Testing

Testing is a very important part of development. It ensures you that everything works as you expect. There exist many different testing techniques. Among all these techniques I have chosen to regularly perform automated tests, assembly tests, regression tests and manual tests. Assembly tests and manual testing go hand in hand in the testing of my application. So if I do any mistake in my code while manipulating data and afterwards I plot a graph above these data, the graph does not visualize data correctly. So I perform some manual testing, it reveals that these data were not correctly manipulated and the code has to be exposed to assembly testing. And vice-versa – when I performed detailed assembly testing, then it was much more probable that my application will pass my manual testing.

4.2.1 Unit testing

Unit testing is performed on the lowest level of the application. The lowest level of my application contains connecting to the database, data manipulation, plotting the charts and plotting the charts based on user input. For creating unit tests I have used R package called **testthat**. The only relevant category out of all the mentioned to be tested is plotting.

4.2.1.1 Charts without user input

Most of the plots in my application have been created via *ggplot2* package. All the charts created via ggplot2 are objects of the same class – **ggplot**. Every instance of class ggplot holds some information. For example, it contains labels of the axes, axes limits, axes breaks, plot layers, input dataset and many more. To perform the unit tests I had to extract information about axes and its data.

Different types of plots required slightly different approach. The following list describes the techniques used for different types of plots:

Bar charts For this type of plots I have extracted data from its ggplot object and summed the counts for each of its bars (columns, or groups) independently. So I got sums of counts for each bar. Then I had to get the counts of all the data of the certain group and compare them. If all these pairs have been equal, the test was successfully passed.

Scatterplots In my application I have unit tested scatterplots with discrete values on x-axis, e.g. x-axis represents transplant year. So the points on scatterplot are literally grouped by these discrete values. On y-axis they have continuous values of x-axis. In this case, I have performed the tests very similarly as in the previous case. I got sums of counts of each discrete x value from ggplot object. Then I got the count of all the data of the certain discrete x value and compared them. Equality of these 2 numbers meant successful test pass.

4.2.1.2 Charts with user input

At this group of charts I have followed the principles of testing described in the section 4.2.1.1. Moreover, I had to simulate the user input. To achieve this I have enwrapped the whole test scenario with for loop. I have iterated over this test scenario multiple times and in each iteration I have randomly generated legit example of user input and tested the plots generated based on this simulated user input.

4.2.2 Manual testing

First, I need to point that this kind of testing is very arduous, difficult and, in my opinion, not that accurate – a human being may easily forget something or just make a mistake. Though I have just mentioned how hard it is to test the application manually, I had to choose this kind of testing for certain scenarios.

Development of the application was backed with manual testing from the very start to the end of the whole development process. Every single chart I had created had to undergo the manual testing. After performing any serious change in my code I started the application and performed manual regression

testing.

Charts testing
Not only I have to be sure that the objects representing the charts really contain the correct data to be visualized but I also have to be sure that these objects really visualize them the way I want them to be visualized. To achieve this I had to compare the output images of charts with the data contained in the objects that represented these charts. Figure 4.1 captures my screen layout during this kind of testing.

Controlling the process of integration
Integration of the external application was a big chapter of my work. Since I was integrating application created by someone else, I wanted it working the same way with my data as it was working in its original state. I had to do a couple of changes in the code of this application. With these changes, definitely, came problems. After each little changes in the code, I had to manually check if the original functionalities of the application still work the same way as they did before applying the changes. With this kind of testing, I have achieved easy-going integration. The only problem I have captured while testing was caused by bad deletion of code. Because of that bad deletion, the application stopped working and announced an error message. This was fixed very easily because I have noticed the problem very early.

Figure 4.1: Example of manual testing

4.2.3 Shinytest

"Shiny developers now have tools for automated testing of complete applications, with the Shinytest package, so that you can be confident that your applications will keep operating as expected." [1]

When I decided to jump into integrating the external application, I was forced to test the application every time I deleted some small piece of code. First, I was doing this testing manually but then I came across Shinytest package. This package seemed to me very handy. It had exactly the same features as I expected a testing package to have for this kind of testing. Although, in the end, I have stuck to manual testing. I have to describe why testing style of this package seems to me so fascinating, why I wanted to use it, and why I could not have used it.

How Shinytest works
Shinytest is probably the ideal R package for testing Shiny applications when it comes to regression testing and simulating user actions. Its efficiency lays in easy creating of tests. Creating tests with Shinytest works in the following way:

1. Start the testing environment by running `recordTest()` function with appropriate parameters.

2. Now the application is running in test mode and you can record the steps that user may perform – set inputs to specific values, submit forms et cetera. When you get your application into a certain state, you want to capture this state. So you tell Shinytest to capture the actual state of the application. When you want to finish creating a test scenario, you press the button to stop recording the test and Shinytest will create some files that represent this test scenario. These files are: 1 JSON file that holds information about inputs and outputs, the PNG file(s) which capture(s) how the application's GUI looks like and the R script that holds the lines of code which represent the sequence of steps that you have just recorded.

3. To perform the recorded test, run `testApp()` with required arguments.

4. R will respond you with a result of a test. If something goes wrong, Shinytest let you check the differences between the actual state of the application and the one captured when you were creating the test.

4.2.3.1 Encountered problem

Although it had seemed to me like a perfect automated testing tool simulating user actions, I was not able to use it. Despite many encountered problems I

have been trying to solve them. I have contacted one of the developers of the Shinytest package and he has managed to fix some issues in the package. Some other issues have remained unfixed yet.

4.2.4 Acceptance testing

Acceptance tests are tests on the customer side. If software passes all the tests on the side of a distributor, it can be delivered to the customer. Mostly, the customer performs acceptance tests with his team of testers. The inconsistencies between the application and the specification are reported to the development team. The development team fixes the problems and the process repeats.[7]

My application has undergone acceptance testing at my customer many times. From time to time I have contacted my customer, told him that certain part of the application is done and I am not planning to change it anymore till he finds the inconsistencies. So I have delivered the application to the client, he run the application, performed the testing and contacted me back if he is fine with the state of the application. If something did not go according to the customer's ideas, I was prepared to fix it. Basically, these tests have represented the ends of the iterations.

I have discovered that the acceptance testing and the iterative software development is a great way to slowly but surely move forward in development. These 2 has gone hand in hand during the whole development process and I must say that it has been the strongest testing style in the whole development process.

Finally, the ÚHKT staff has reviewed the final state of the application and have considered that the application is in a great state and it can be deployed.

4.2.4.1 Tab Frequency tables

Already mentioned part of my application is the Frequency table tab. The client has checked my solution and found out that another solution – the external application – would fit better for this tab. The client has tried out my solution, compared it with external application and he has decided for the external application because it fits him better for his kind of work. All the other details why my client has decided to move from the original frequency tables to the external application are described in 3.1.4.4

4.3 Deployment

The actual state of the application has been successfully deployed in ÚHKT. Any problems have not occurred during the deployment and everything has run smoothly. So the application is running over ÚHKT's internal network now and can be used by users.

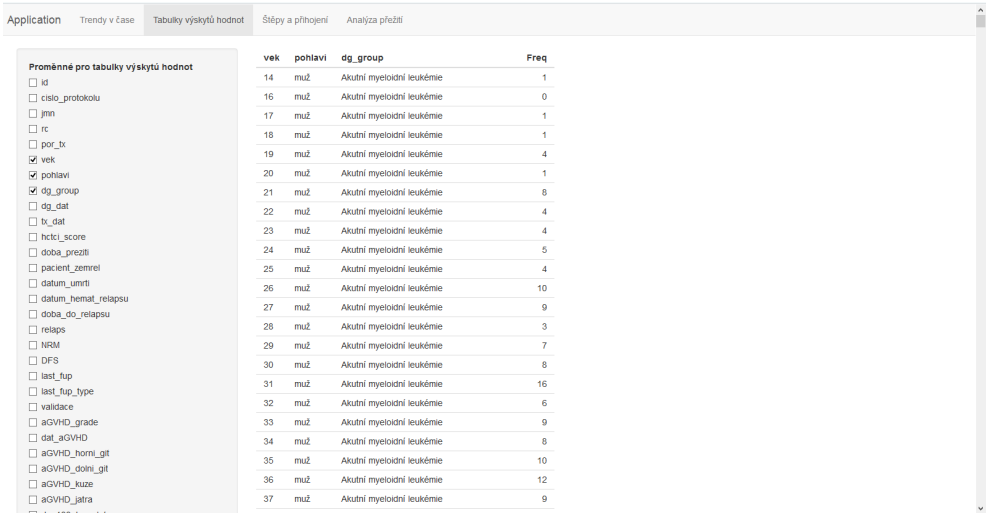

Figure 4.2: Frequency tables tab before the acceptance test

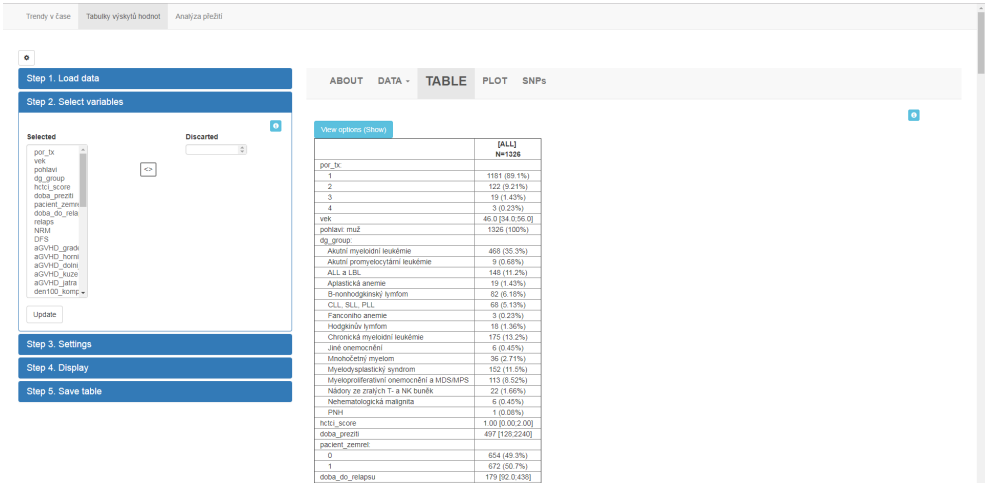

Figure 4.3: Frequency tables tab after the acceptance test

Conclusion

Summary

The main goal of my thesis was to design and implement a web application that is meant to visualize patients data.

The chapter *Theoretical part* gathers all the information about plotting. It names and describes all types of plots used in my application.

In the first part of software development, I have analysed the problem with my client and I have written down all the things needed for proper analysis. I have put down information about existing solutions, then I analysed client's requirements and therefore I have analysed the key technologies of my project.

The design of the application consists of the very detailed description of my dataset. I have described all the variables of my dataset that I work with. Next, it was needed to design the GUI so the written description combined with the wireframes have done the job. Then, in the section *Plotting*, I thoroughly described all the plots that I have created in my application. Each plot has its name, its type (Bar Chart, Scatter Plot,...), its variables and also it may contain user input panel. The last section of the application's design is *Directory layout*. I have created a logical directory structure here.

The chapter named *Realisation* consists of *Implementation* section and *Testing* section. The first one, *Implementation*, tells few words about my steps applied while implementing the application and about the packages that have helped me to build the code and the application the way I wanted it to be built. Testing describes the testing methods I have applied. I am happy for this part because it helped me to speed up the application and to detect bugs. In the end, the ÚHKT staff has successfully deployed the actual state of the application on their server so it is running over their internal network now.

Now I hope that this application will help ÚHKT staff in any way and brings them value.

Future development

The actual state of the application is the foundation of a big project. This state allows the ÚHKT staff to observe the basic statistics. Their plan was to let me build the essential fundamentals, create the software's architecture and later they will build the application further on their own.

At this moment I know that ÚHKT is planning to add another tab in the near future and it will be named **Grafts and engraftment**. I have already prepared basic files for the code of server logic and the UI of this tab. This tab will use the already known dataset from tab Trends in time.

Bibliography

[1] RStudio. *Testing Shiny applications with Shinytest* [online]. [cit. 2018-03-26]. In: `https://www.rstudio.com/resources/webinars/Testing-Shiny-applications-with-Shinytest/`.

[2] Wikipedia contributors. *Plot (graphics)* [online]. 2 August 2017 07:44 UTC [cit. 2018-04-08]. Available from: `https://en.wikipedia.org/w/index.php?title=Plot_(graphics)&oldid=793517812`.

[3] MLEJNEK, Jiří. 3. Analýza a sběr požadavků - případy užití. In: *BI-SI1 Softwarové inženýrství I.* [online]. [cit. 2018-04-18]. Available from: `https://edux.fit.cvut.cz/courses/BI-SI1/_media/lectures/03/03.prednaska.pdf`.

[4] Hematopoietic Cell Transplantation-Comorbidity Index (HCT-CI). In: *HCT-CI Calculator Web site* [online]. Fred Hutchinson Cancer Research Center. [cit. 2018-04-17]. Available from: `http://www.hctci.org/`.

[5] WICKHAM, Hadley. *ggplot2.* [online]. 2013 [cit. 2018-04-22]. Available from: `http://ggplot2.org/`.

[6] SUBIRANA, Isaac, HÉCTOR, Sanz, VILA, Joan. Building Bivariate Tables: The compareGroups Package for R. In: *Journal of Statistical Software* [software]. 2014, 57(12), 1–16. [cit. 2018-04-23]. Available from: `http://www.jstatsoft.org/v57/i12/`.

[7] HLAVA, Tomáš. Druhy, typy a kategorie testů. In: *Testování softwaru* [online]. 2011. [cit. 2018-03-25]. Available from: `http://www.testovanisoftwaru.cz/tag/akceptacni-testovani/`.

[8] IRIZARRY, Rafael, LOVE, Michael. dplyr tutorial. In: *PH525x series - Biomedical Data Science* [online]. [cit. 2018-04-27]. Available from: `http://genomicsclass.github.io/book/pages/dplyr_tutorial.html`.

[9] CHANG, Winston. App formats and launching apps. In: *Shiny from RStudio* [online]. RStudio Inc., 2017. [cit. 2018-05-12]. Available from: `https://shiny.rstudio.com/articles/app-formats.html`.

Appendix A

Acronyms

CSS Cascading Style Sheets

GUI Graphical user interface

HCT Hematopoietic Cell Transplantation

HCT-CI Hematopoietic Cell Transplantation-Comorbidity Index

HLA Human leukocyte antigen

HTML Hypertext Markup Language

IDE Integrated Development Environment

ID Identification

JSON JavaScript Object Notation

MUD Matched unrelated donor

PBPC Peripheral blood progenitor cells

PDF Portable Document Format

UI User interface

ÚHKT Ústav hematologie a krevní transfuze (Institute of Hematology and Blood Transfusion)

Appendix B

My outputs

The following text and images display and slightly describe the outputs created by my application. All the plots are described in detail in the section 3.2.3 and the variables of these outputs are described in the section 3.2.1.

All the outputs are exposed here except 1, **Conditioning regimens used by year**. This one is too large to be placed here so I will omit this one and will not show it here. Anyway, basically this plot represents the same information as **Conditioning regimens** plot does.

Transplantations by year
This plot displays the numbers of transplantations performed by ÚHKT in each year. In the introduction of the thesis, I have mentioned that ÚHKT performs tens of transplantations per year. This plot proves this statement.

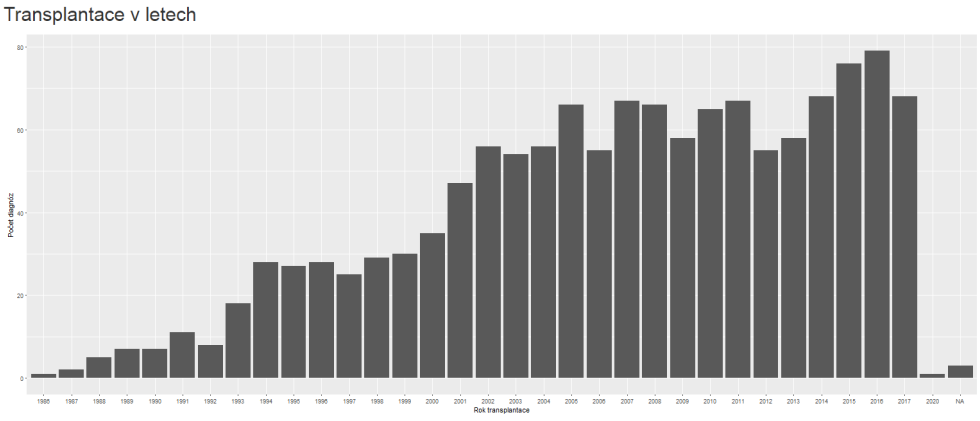

Figure B.1: Transplantations by year

Transplantations by diagnosis
This output of my application shows that the diagnosis with the highest occurrence in the whole history of ÚHKT's data collection is acute myeloid leukemia.

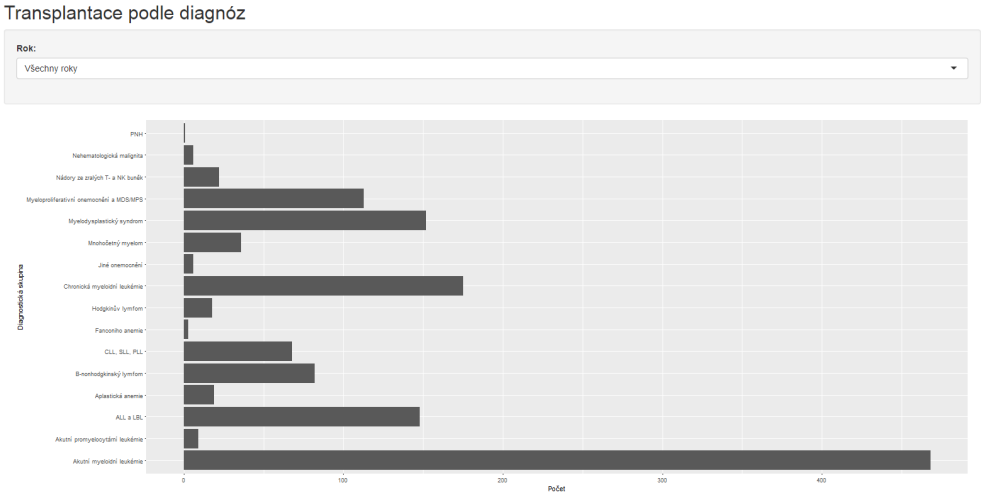

Figure B.2: Transplantations by diagnosis

Transplant type
This output from my application displays the counts of transplant types performed in ÚHKT in the last 6 years. You can see that MUD (matched unrelated donor) transplant is the most frequently performed transplant with its count over 150 patients. At this kind of transplant, cells for a donee are not provided by a close family member but by a donor from the general public.

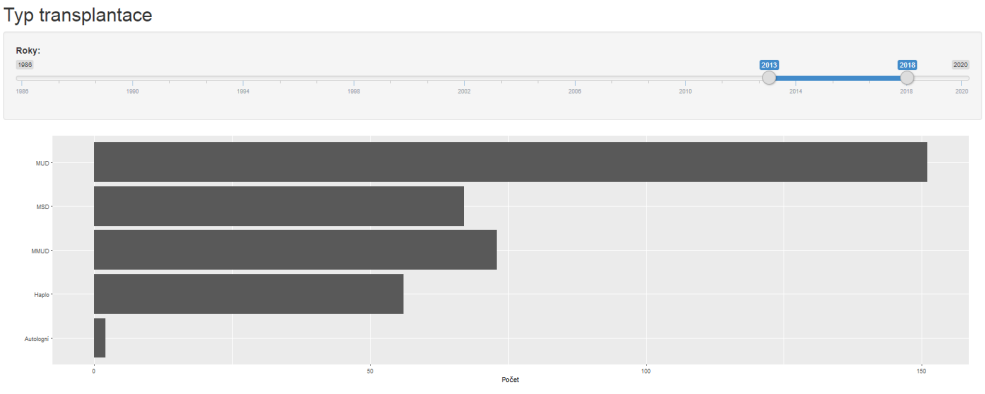

Figure B.3: Transplant type

Indications for HCT by year

This plot displays to a user the counts of transplants by diagnosis separated by years. The image displays the time period of last the 6 years. Acute myeloid leukemia dominates again.

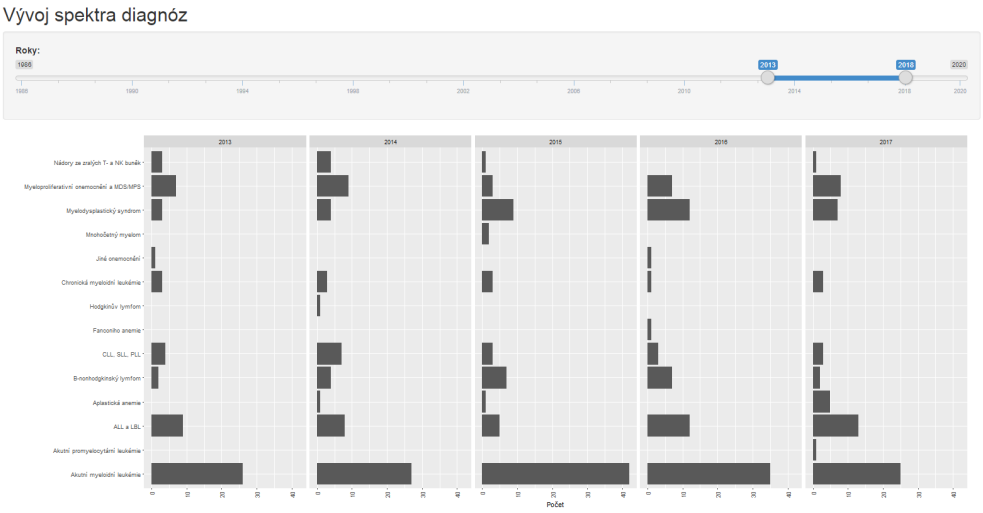

Figure B.4: Indications for HCT by year

Transplant type by year

This plot is the stacked bar chart displaying the ratios of transplant types performed in single years.

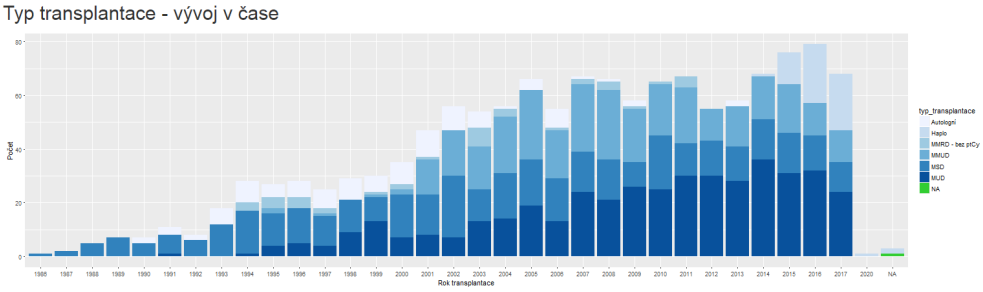

Figure B.5: Transplant type by year

Type of graft
The Type of graft output displays the frequencies of the values of the variable *transplantace.typ_stepu*. The most dominating type of graft is *PBPC*. PBPC graft says that the hematopoietic cells are gotten from a donor by an apheresis machine.

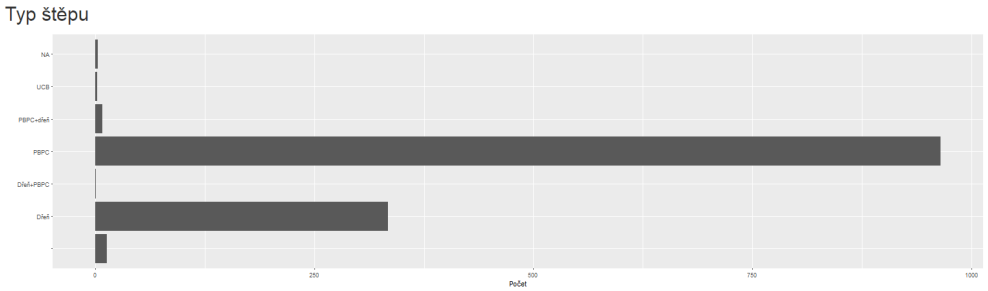

Figure B.6: Type of graft

Conditioning regimens
This plot displays the counts of different types of conditioning regimens performed in the year 2017. I am not able to describe the plot in detail because it requires deep knowledge in the field of haematology.

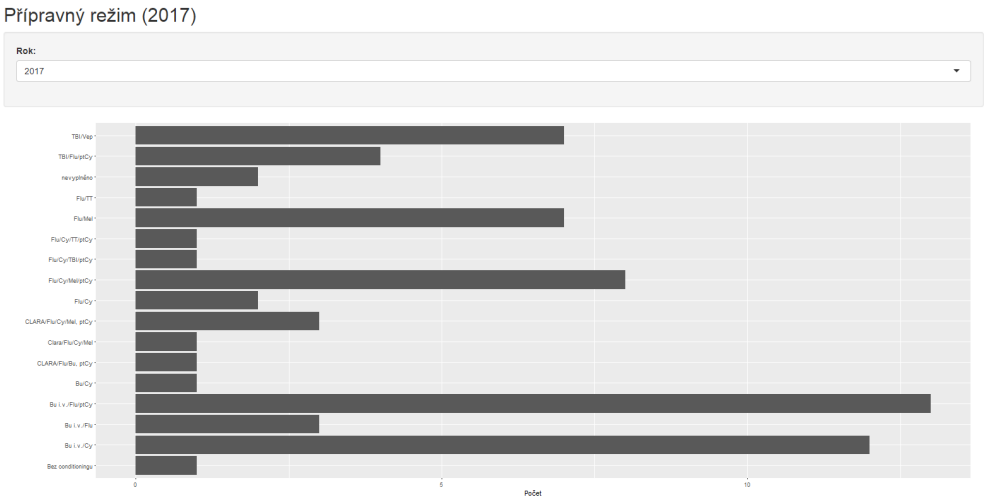

Figure B.7: Conditioning regimens

Age of the patients

This plot displays age of the patients in single years and the medians of the age in each year drawn with the red coloured dot. Mostly the patients are from the age group from 20 to 68 years. The median is mostly around 45 to 55 years.

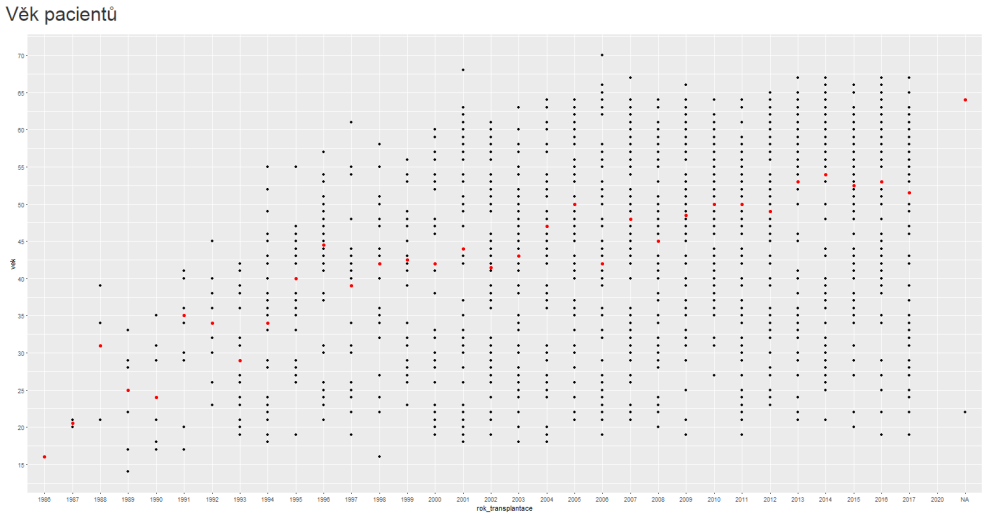

Figure B.8: Age of the patients

HCTCI zones

Every patient has his HCT-CI score. ÚHKT divides the values of HCT-CI scores into 3 categories (zones) – 0, 1-2, >2. This stacked bar chart displays the ratios of these categories per years.

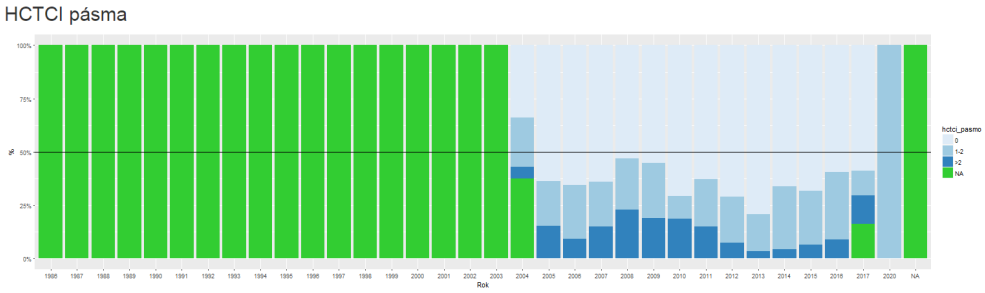

Figure B.9: HCTCI zones

Sex of the patients by year
This plot is also the stacked bar chart but this time the counts are expressed in percentages. It shows the ratios of men to women who undergo transplants. The ratios are expressed per years. The dataset is anonymised so the chart displays 100% of men in each year.

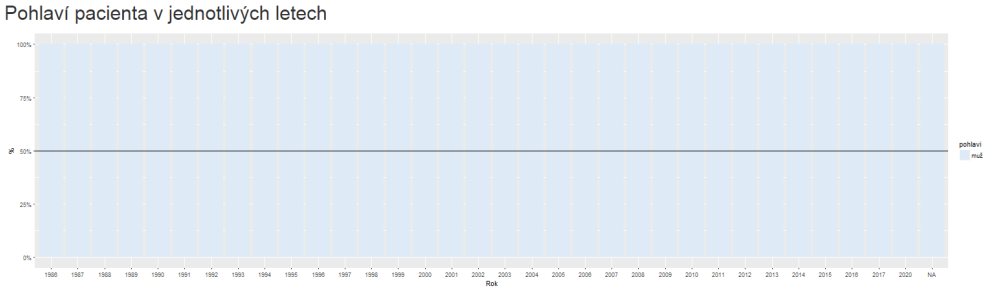

Figure B.10: Sex of the patients by year

Survival analysis
This plot displays the survival curves. x-axis represents number of survived months and y-axis says how many patients has survived (in percentages). The curves are grouped by 5 year intervals from the year 1986 to the year 2020. You can see that around 40% to 50% of the patients live in the long terms after the transplantation.

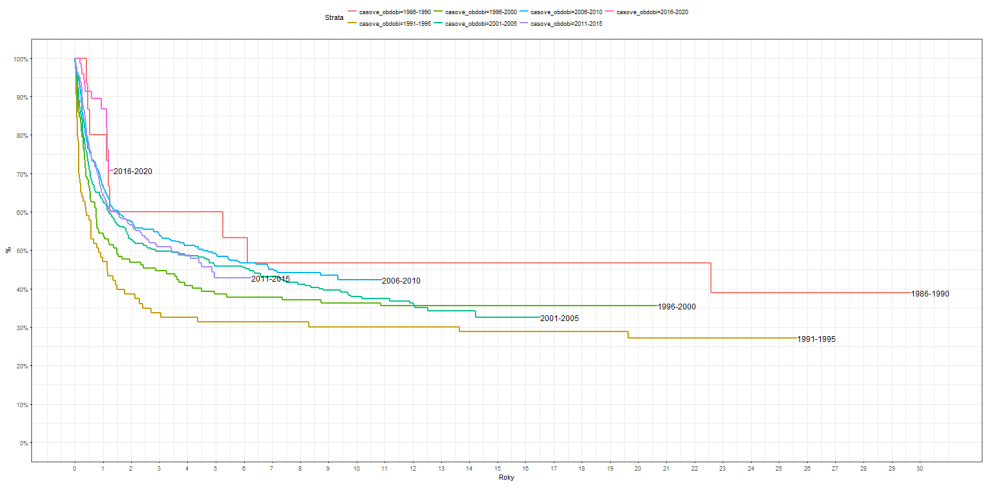

Figure B.11: Survival analysis

Appendix C

Installation guide

I have written the installation guide in Slovak language. That is because the application will be used by the Czech hospital and therefore it is not necessary to write it in English. The following is the text of the installation guide.

Krok č. 1
V prvom rade je potrebné, aby mal užívateľ nainštalovaný interpreter pre R. Ten si môže stiahnuť z následujúcej stránky – `https://cran.r-project.org/mirrors.html`.

Krok č. 2
Je treba správne nastaviť konštanty pre pripojenie k MySQL databázi a SQL dotaz pre správne kódovanie znakov pri databázových transakciách.

V súbore **src/db/db_connection.R** si užívateľ nastaví konštanty **host**, **port**, **dbname**, **user**, **password** podľa svojich potrieb.

V súbore **src/db/query.R** sa nachádza konštanta **set_names_query**. Táto konštanta obsahuje string s MySQL dotazom, ktorý nastaví kódovanie znakov pre všetky databázové transakcie, ktoré aplikácia využíva. Podľa toho, aké kódovanie znakov používa užívateľov R intrepeter, je nutné nastaviť vhodný character set.

Krok č. 3
Keď už má užívateľ nainštalovaný interpreter pre R, tak ho spustí a otvorí sa mu konzola. Teraz užívateľ spustí následujúce príkazy:

```
library(shiny)  # load shiny library
# change application_folder to your application directory
application_folder <- "C:/Users/Test/app/"
# change app_port to whatever free port - app will be
# listening on this port
app_port <- 8080
runApp(application_folder, app_port)
```

Kód si treba prispôsobiť podľa vlastnej potreby. Do premennej `application_folder` musí užívateľ priradiť cestu k aplikácii – absolútna cesta zložky, v ktorej sa nachádza súbor **app.R**. Do premennej `app_port` je treba priradiť číslo portu, na ktorom užívateľ chce, aby mu bežala aplikácia.

Krok č. 4
Ak má užívateľ nainštalované všetky balíčky, ktoré aplikácia využíva, tak sa mu aplikácia úspešne spustí. Ak užívateľ nemá nainštalované balíčky, tak ho konzola upozorní na to, že potrebuje nainštalovať určité balíčky. Keď úžívateľ nainštaluje všetky potrebné balíčky a zopakuje **Krok č. 3**, tak mu aplíkacia pobeží. Výsledkom bude, že z užívateľovho počítaču spraví balíček **Shiny** server a ostatní užívatelia budú môcť navštíviť stránky, ktoré aplikácia poskytuje.

Appendix D

Contents of enclosed CD

- `readme.txt` ..... the file with CD contents description
- `instalacna_prirucka.pdf` ..... installation guide
- `src` ..... the directory of source codes
  - `impl` ..... the source codes of implementation
  - `thesis` ..... the directory of LaTeX source codes of the thesis
- `text` ..... the thesis text directory
  - `thesis.pdf` ..... the thesis text in PDF format